

Построение современных lakehouse архитектур с помощью Presto

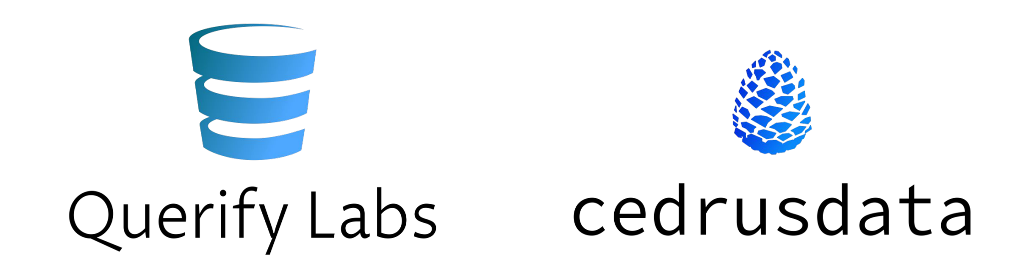
Владимир Озеров
Querify Labs



План

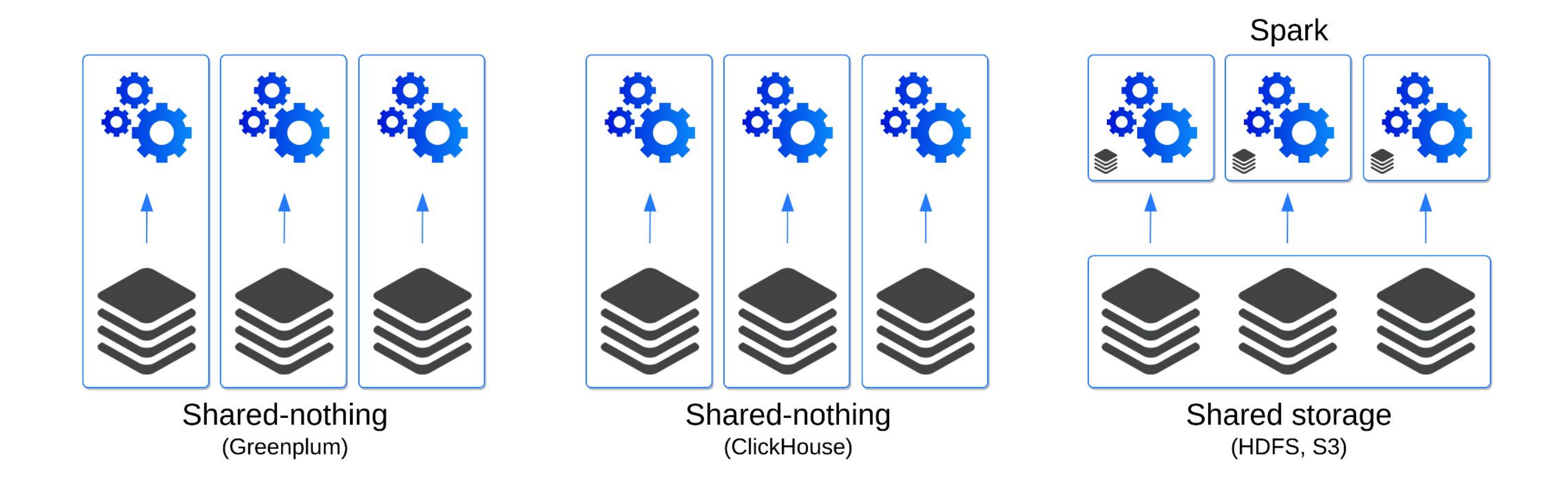
- Что такое **Lakehouse**?
- Что такое **Presto / Trino**, как он устроен внутри, и при чем тут lakehouse?

Querify Labs



- Разработка новых аналитических СУБД и data-management систем для технологических стартапов по всему миру (стек — Java/C++, Apache Calcite, Apache Arrow, Velox).
- Разработка российской аналитической платформы CedrusData на основе Trino.
- Контрибьютим в Apache Calcite и Apache Arrow.

Аналитический стек



Data Warehouse — real-time/ad-hoc/interactive analytics, compute и storage совмещены.
Data Lake — batch processing, compute и storage разделены.

Проблемы

Data Warehouse:

- Дублирование данных в проприетарных форматах.
- Изменение топологии влияет на **доступность данных**. Отсюда сложности с масштабированием и переходом в облако.
- Не всегда просто бъединять данные между различными системами.
- ETL.

Data Lake:

- Нет транзакций.
- Нет schema enforcement.
- Медленно (?)

Что мы хотим?

- Данные в дешевом хранилище с минимальным дублированием.
- Легкое масштабирование в облаке и on-premise.
- Более высокое качество данных за счет транзакций и schema enforcement.
- Не только batch processing, но и интерактивная аналитика.
- Возможность анализировать как можно больше данных через через единый интерфейс.
- Доступ через SQL.

Эволюция Data Lake

Map-Reduce (Hive), Spark

Stage 1 - - -> Disk - - -> Stage 2

Hive Metastore

Метаданные

ORC | Parquet
Avro | Raw

Storage
(S3, HDFS, local)

Преимущества:

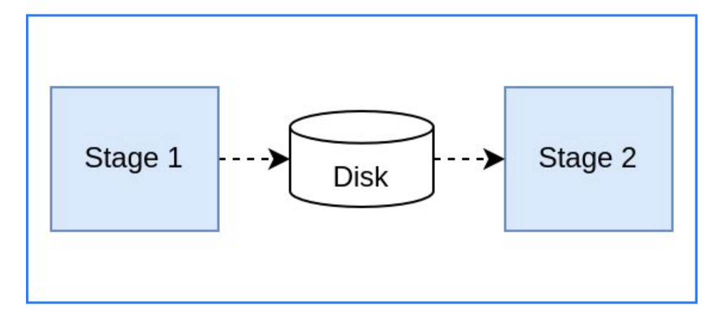
- Storage и compute масштабируются независимо.
- Данные хранятся в открытых форматах.
- Есть поддержка SQL.

Проблемы:

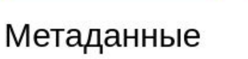
- Map-reduce и Spark не являются оптимальным решением для интерактивной аналитики.
- Нет транзакций и schema enforcement.
- Hive Metastore может быть узким местом.

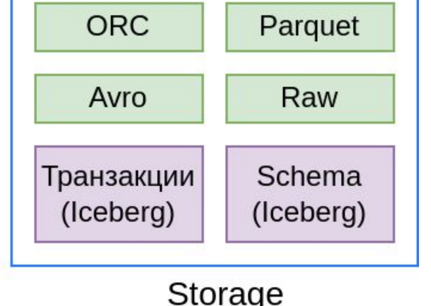
Эволюция Data Lake: Apache Iceberg

Map-Reduce (Hive), Spark



Hive Metastore





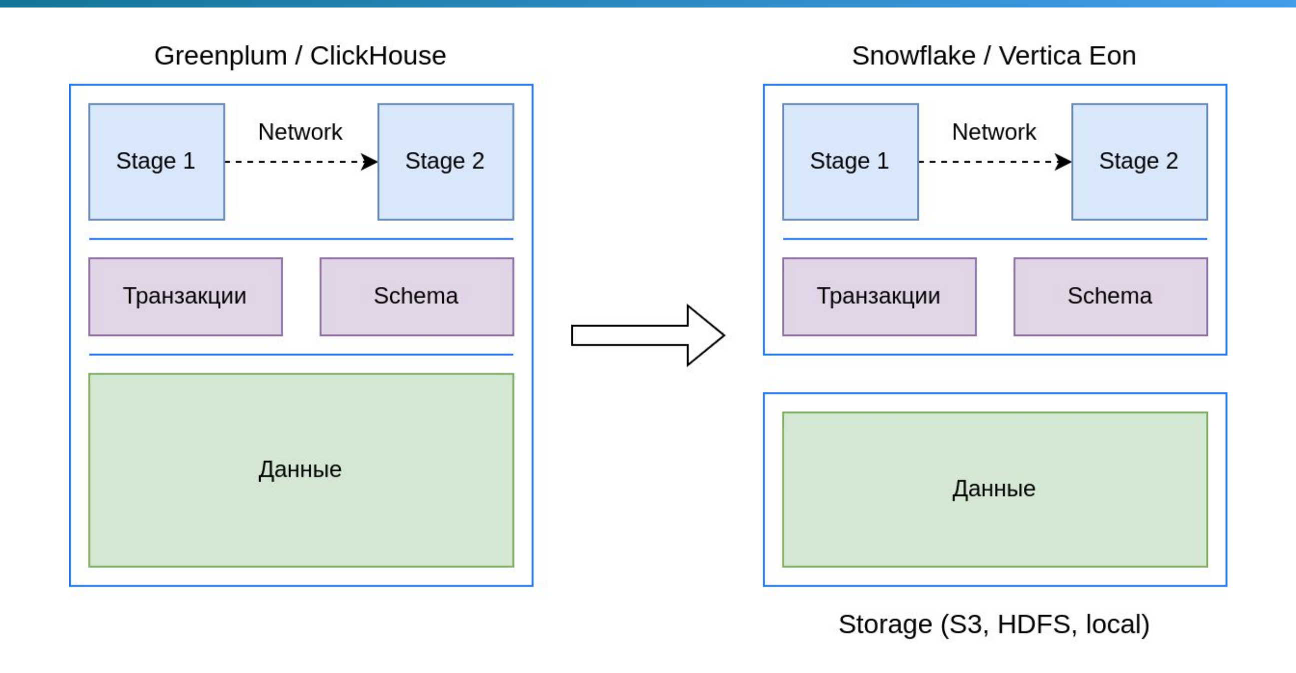
Storage
(S3, HDFS, local)

Apache Iceberg переносит управление транзакциями и schema enforcement на уровень storage, позволяя различным сторонним системам безопасно работать с одними и теми же данными.

Проблемы:

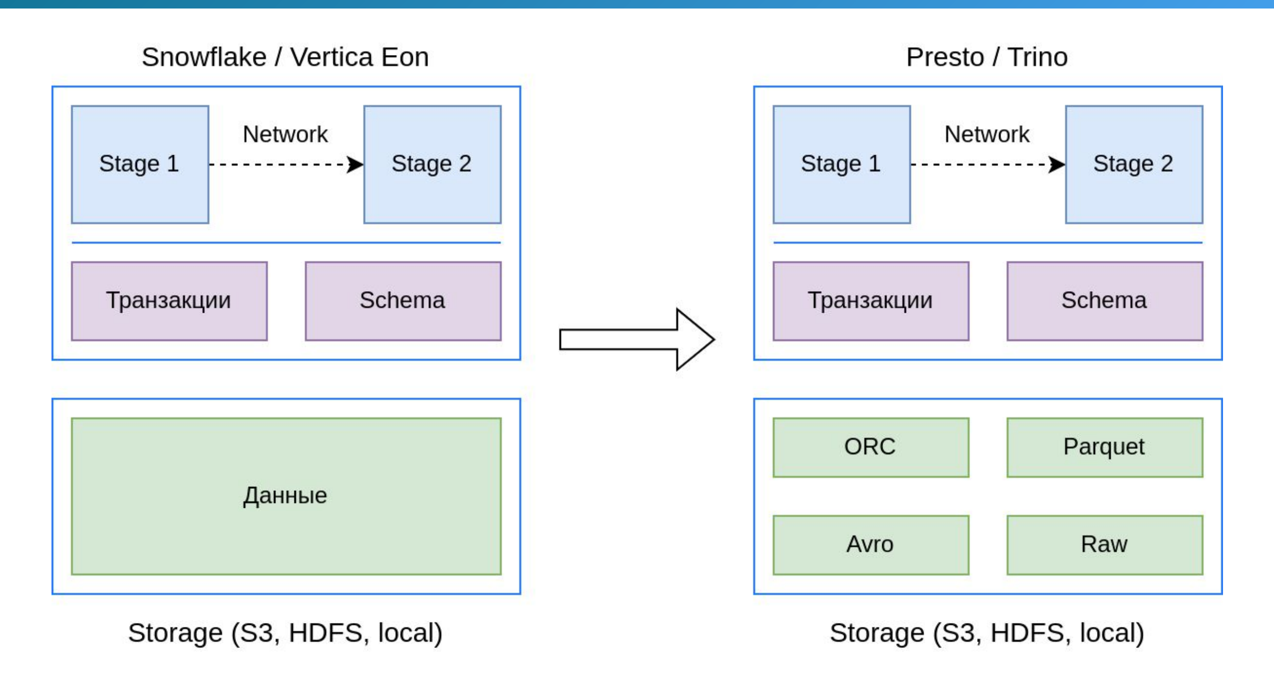
- Map-reduce и Spark не являются оптимальным решением для интерактивной аналитики.
- ~~Нет транзакций и schema enforcement.~~
- ~~Hive Metastore может быть узким местом.~~

Эволюция Data Warehouse: disaggregated storage



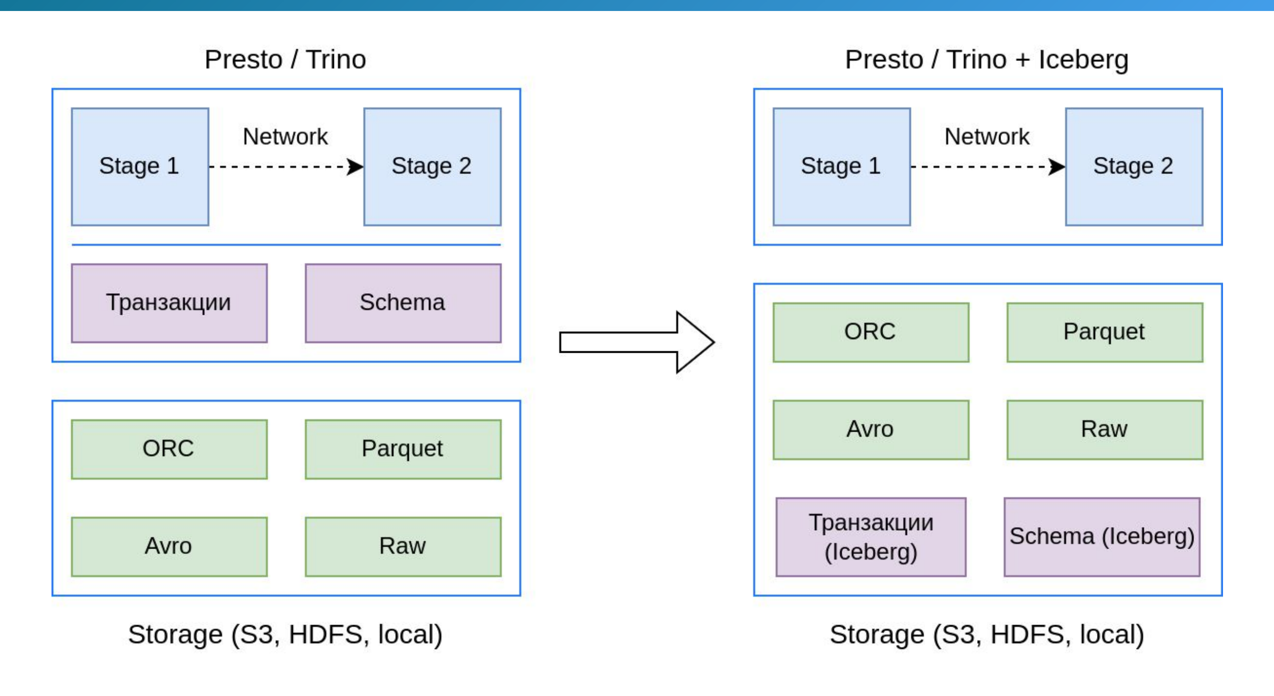
Шаг 1: отделим storage от compute, чтобы масштабировать их независимо.
Драйверы: SSD и быстрые облачные хранилища.

Эволюция Data Warehouse: открытые форматы



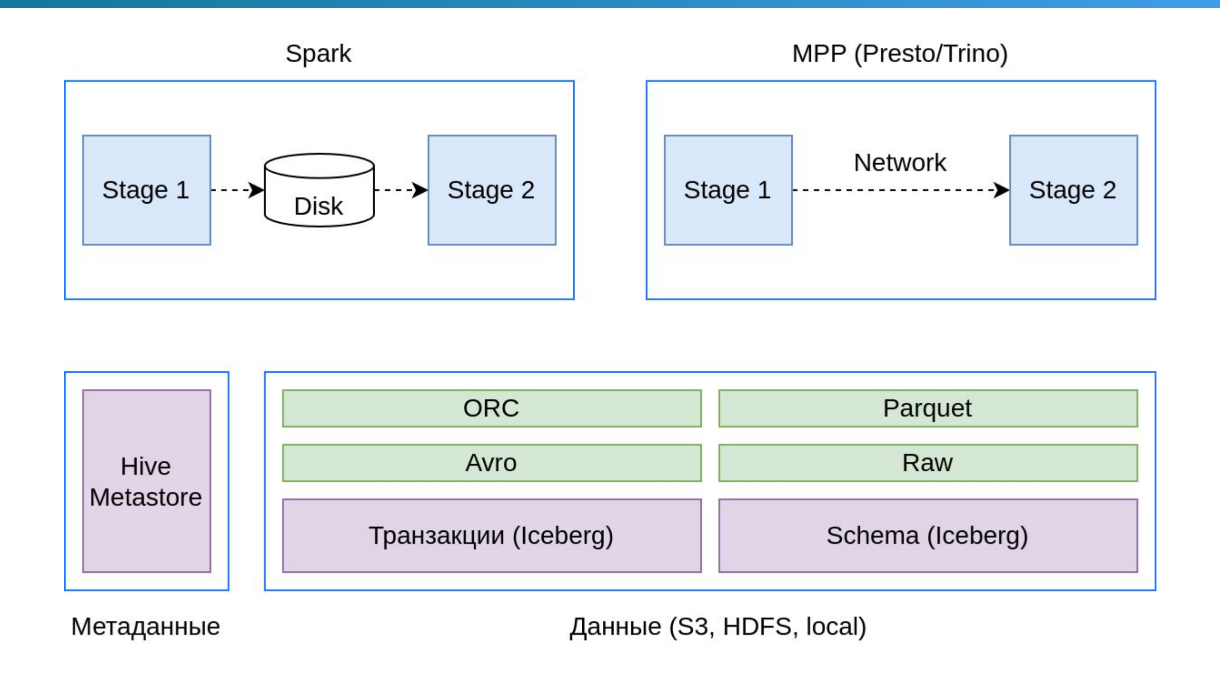
Шаг 2: заменим проприетарные форматы на открытые.
Драйверы: развитие колоночных форматов (Parquet, ORC) и SQL-оптимизаторов (pushdown).

Эволюция Data Warehouse: открытые форматы



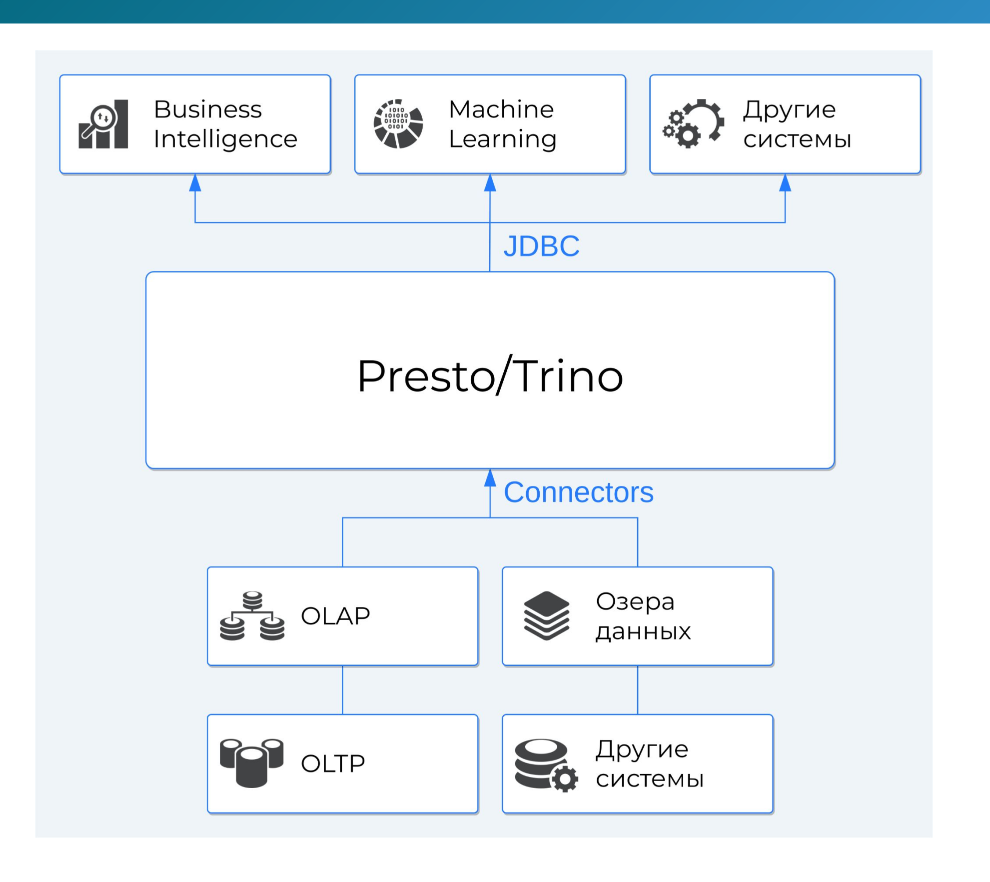
Шаг 3: перенесем управление транзакциями и схемами из СУБД в storage.
Драйверы: Apache Iceberg, Apache Hudi, Delta Lake.

Lakehouse



Результат: множество движков, каждый из которых заточен под определенные сценарии, которые безопасно работают с одними и теми же данными в дешевом распределенном хранилище.

Архитектура Presto



Presto/Trino — это распределенный SQL-движок.

Подключается к источникам данных с помощью коннекторов:

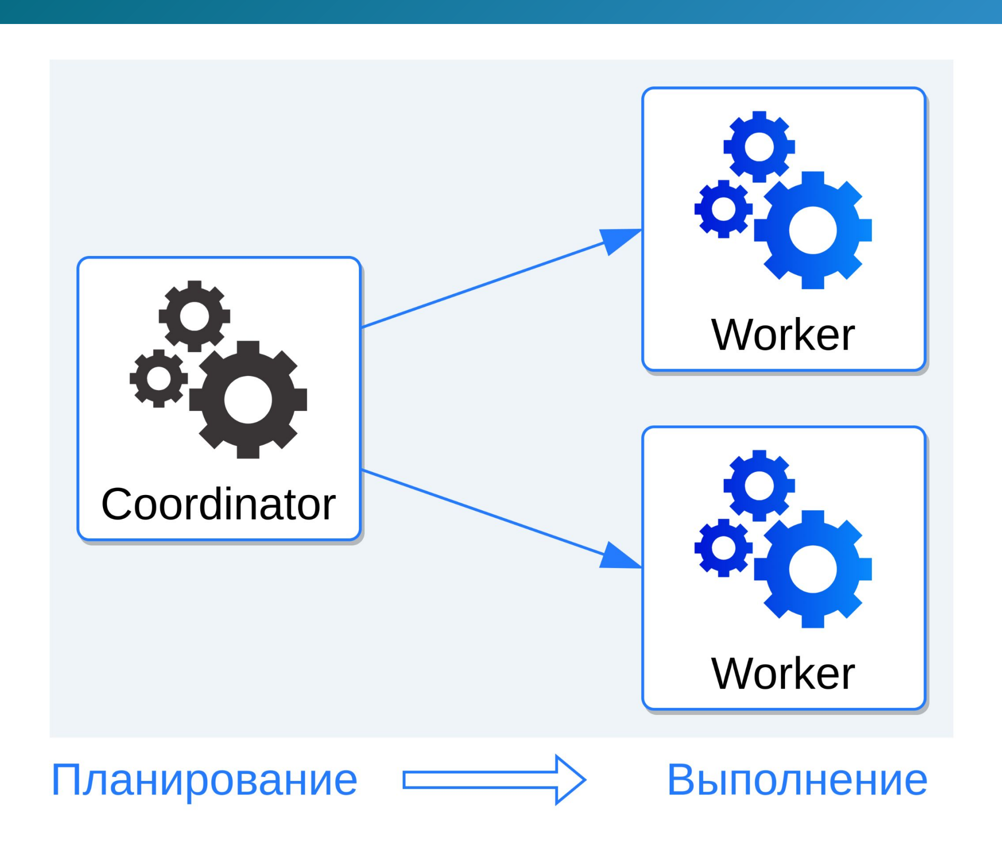
- Озера данных под управлением Hive Metastore и Apache Iceberg.
- Хранилища данных: Greenplum, ClickHouse, Apache Druid, Apache Pinot.
- Реляционные СУБД: Postgres, MySQL, Oracle, SQL Server, MariaDB.
- Нереляционные источники: Cassandra, MongoDB, Redis, Kafka, …

Отдает данные через **JDBC**.

Плагины и коннекторы

- **Plugin** — набор расширений функционала Presto.
- **Connector** — опциональный компонент плагина, который описывает логику работы с источниками данных определенного типа.
    - Пример: Postgres.
- **Catalog** — инстанс коннектора, который работает с конкретным источником.
    - Пример: конкретный инстанс Postgres.

Архитектура: типы узлов



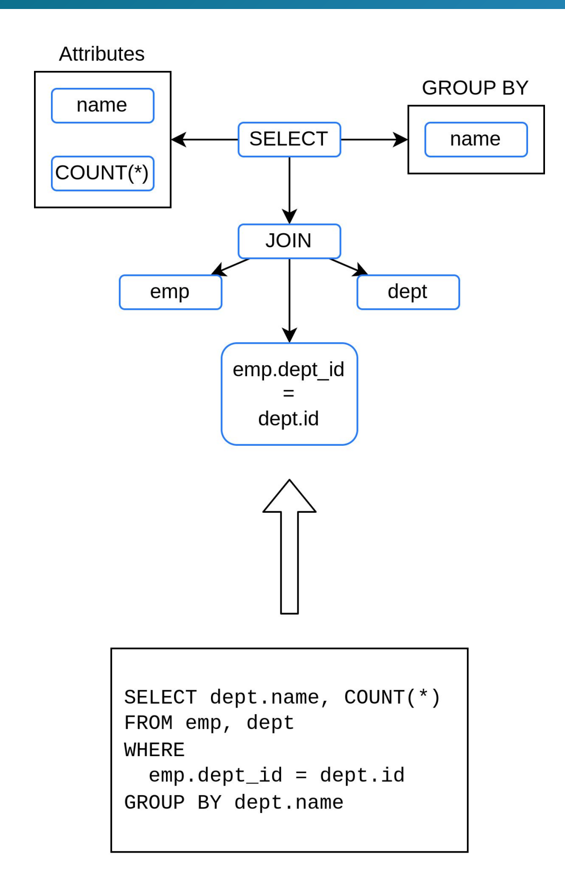
Coordinator:

- Получает, планирует и координирует запросы.
- Может выполнять запросы, если стоит соответствующий флаг.
- При необходимости кластер может содержать несколько координаторов.

Worker:

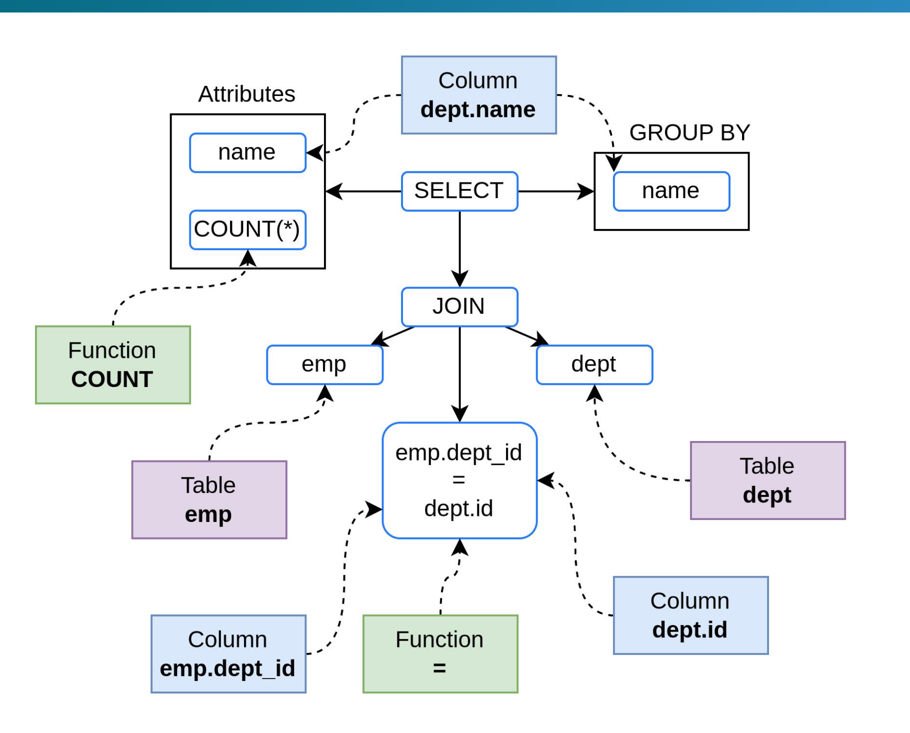
- Выполняет запросы.

Планирование: парсинг



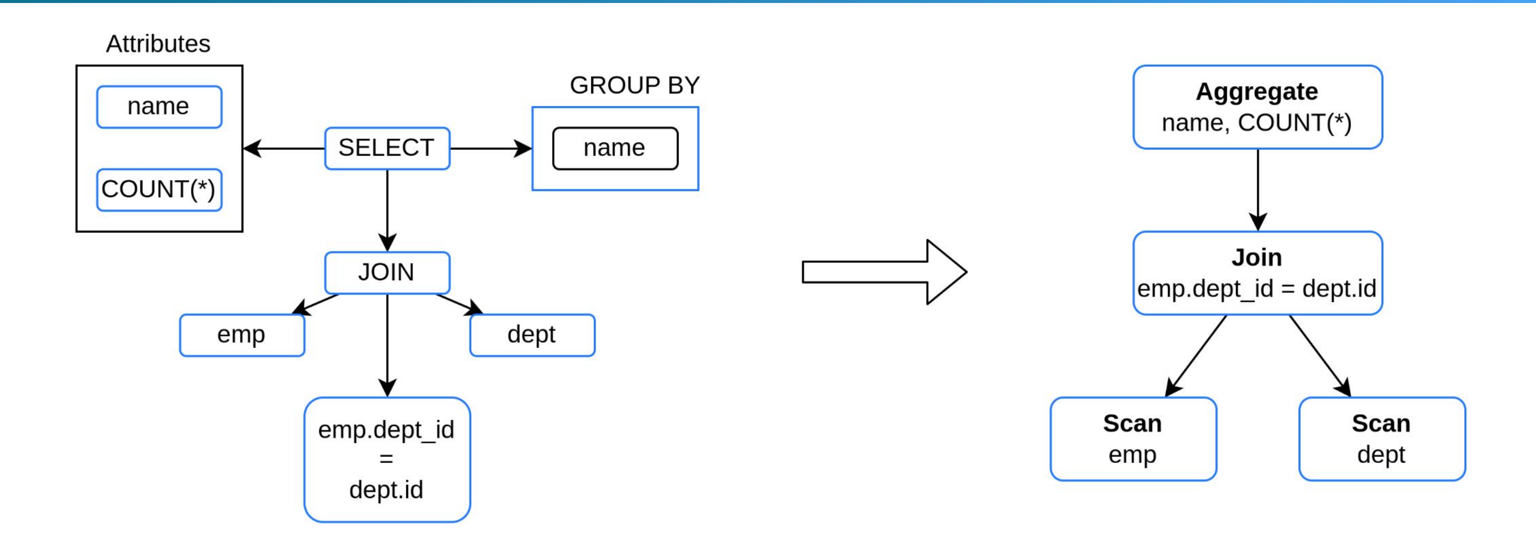
- Задача: превратить SQL-строку в синтаксическое дерево.
- Реализован с помощью ANTLR.
- См. SqlBase.g4.

Планирование: семантическая валидация



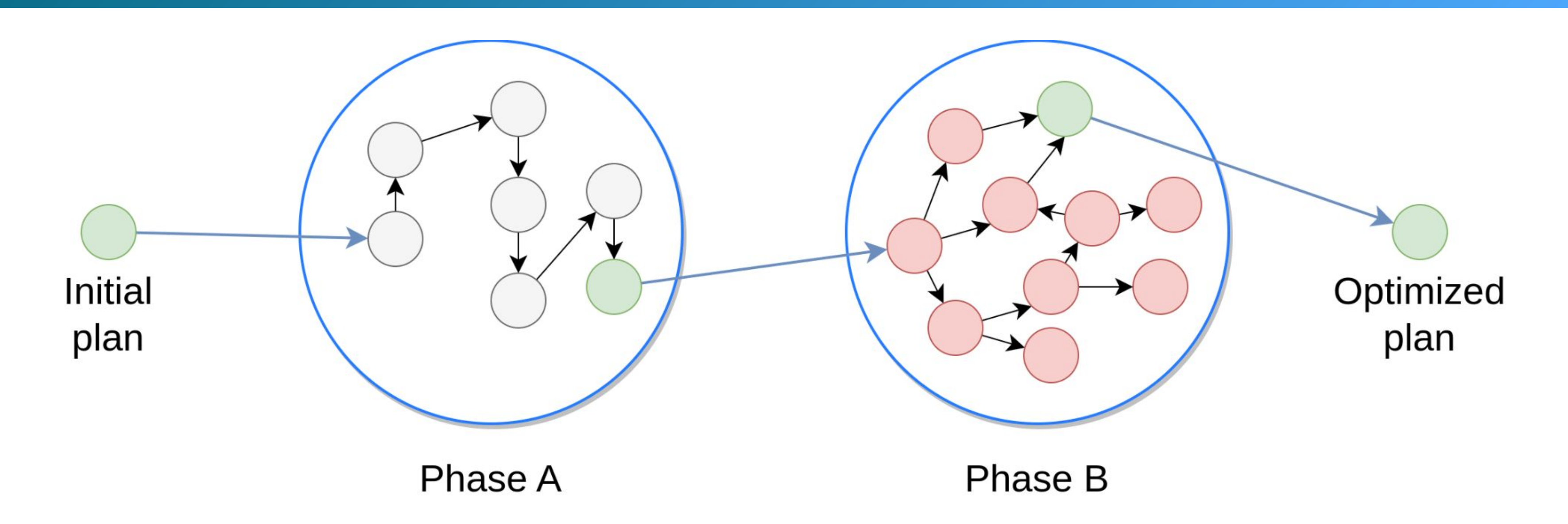
- Задача: определить объекты, участвующие в запросах; убедиться в семантической корректности.
- В отличие от синтаксического анализа, семантический анализ не поддается автоматизации. Реализован большим количеством “спагетти”-кода.
- **Коннекторы** определяют, какие объекты доступны системе (table, column, …).
- **Плагины** могут определять дополнительные функции.
- См. Analyzer.java.

Планирование: трансляция



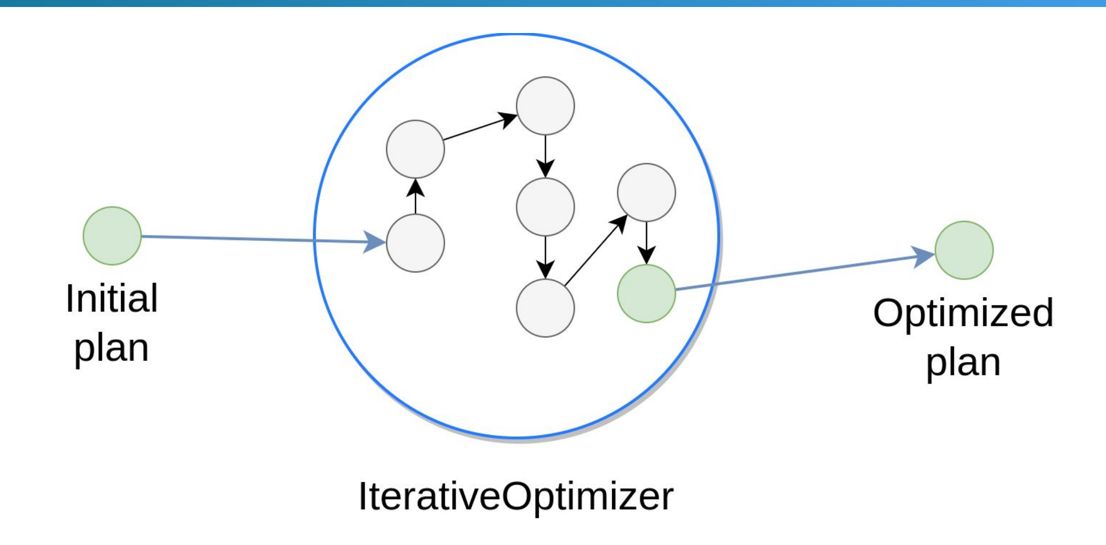
- Современные планировщики SQL-движков чаще всего работают с **реляционным** представлением.
- Presto использует реляционное представление:
  - Scan, Project, Filter, Aggregation, Join, SetOp (Union, Minus, Intersect), …
- См. PlanNode.java, RelationPlanner.java.

Планирование: фазы



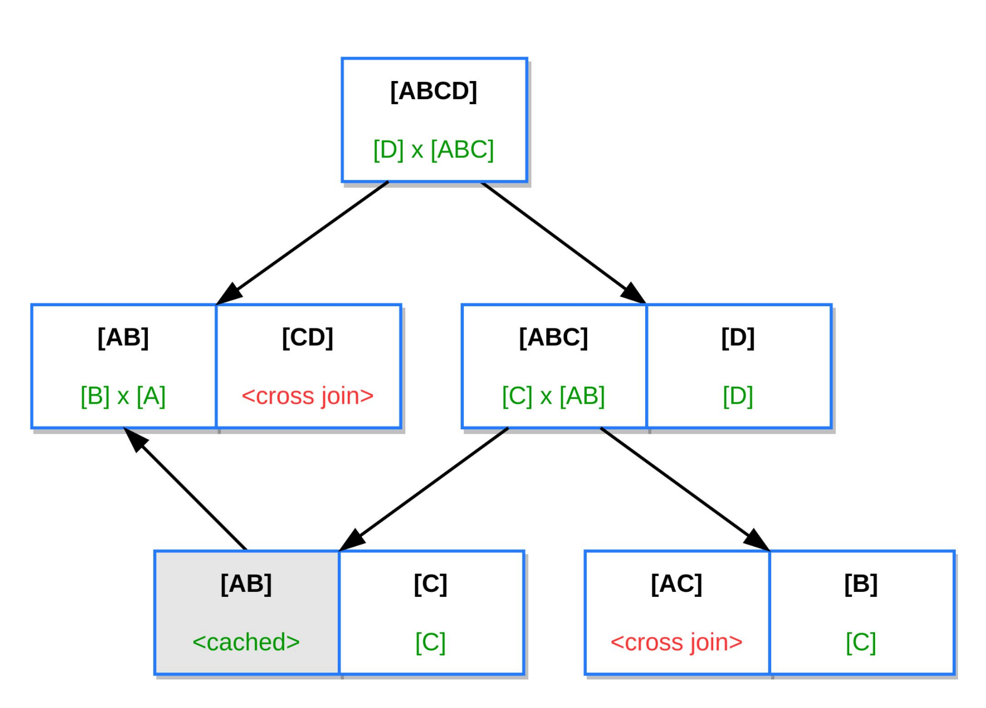
- Планирование организовано в последовательность шагов.
- На каждом шаге мы получаем на вход один план и производим другой.
- Планирование в Presto состоит из примерно 80 шагов.
- См. PlanOptimizer.java, PlanOptimizers.java.

Планирование: итеративный оптимизатор



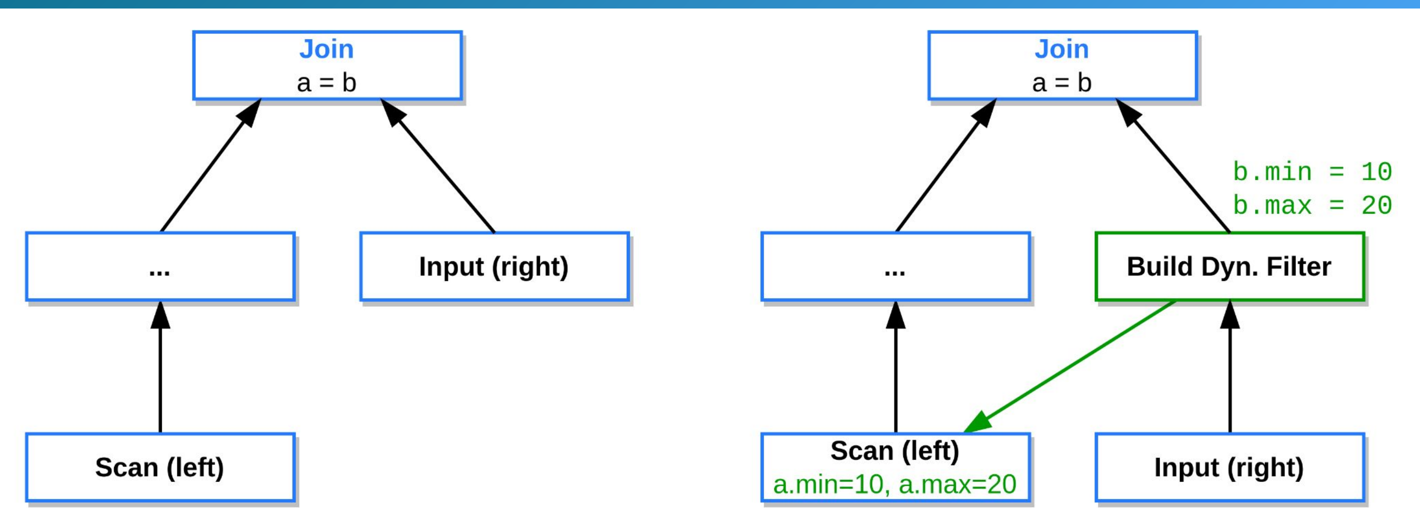
- Оптимизатор принимает начальный план и набор правил, отдает новый план. Работает, пока есть возможность применять правила. Не является cost-based.
- Правило представлено паттерном и логикой трансформации. Примеры: упрощение выражений, filter pushdown, pushdown вычислений в коннектор.
- См. IterativeOptimizer.java.

Планирование: join order



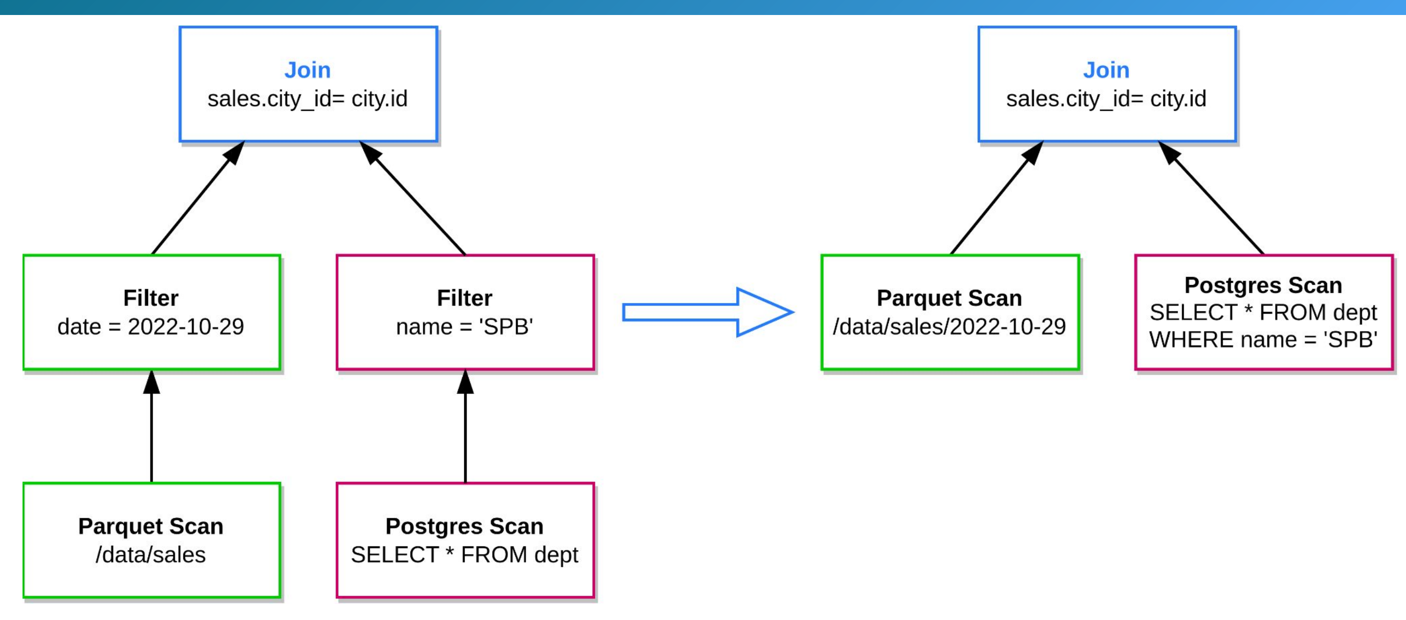
- Класс эквивалентности `[ABCD]` — это все возможные перестановки отношений A, B, C, D.
- Проход вниз: разбиваем классы эквивалентности на более мелкие пары: `[ABCD]->[ABC]x[D]`.
- Проход вверх: находим оптимальный порядок для класса эквивалентности на основании стоимости.
- Стоимость — это функция статистик (row count, min, max, null count, ndv).
- Для Scan-операторов статистики предоставляются коннекторами. Для остальных операторов статистики вычисляются с помощью эвристических формул.
- См. ReorderJoins.java.

Планирование: динамические фильтры



- Идея: посчитать в runtime предикаты одной стороны Join и применить их к другой стороне.
- Наибольший выигрыш происходит за счет pushdown динамического фильтра в Scan.
- В Presto присутствует два типа динамических фильтров:
  - **Локальные** — вычисление и применение происходит на одном узле.
  - **Распределенные** — вычисление на координаторе, применение на воркерах.

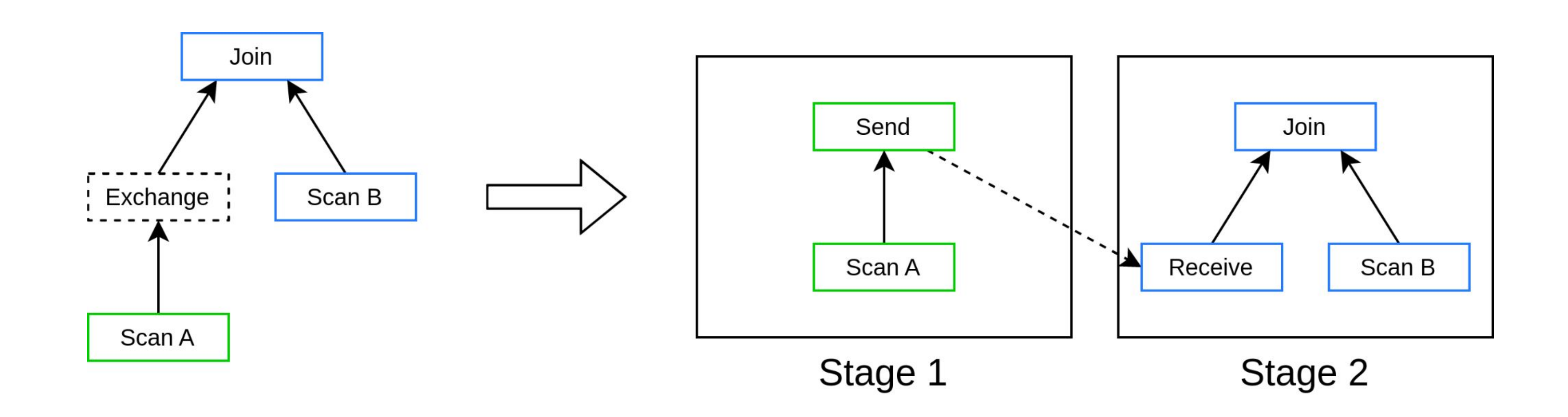
Планирование: connector pushdown



- Коннекторы могут предоставлять свою логику для организации pushdown:
  - **Hive**: filter pushdown, partition pruning, partial aggregation (напр. MIN/MAX).
  - **JDBC**: можно запушить практически все, что угодно.

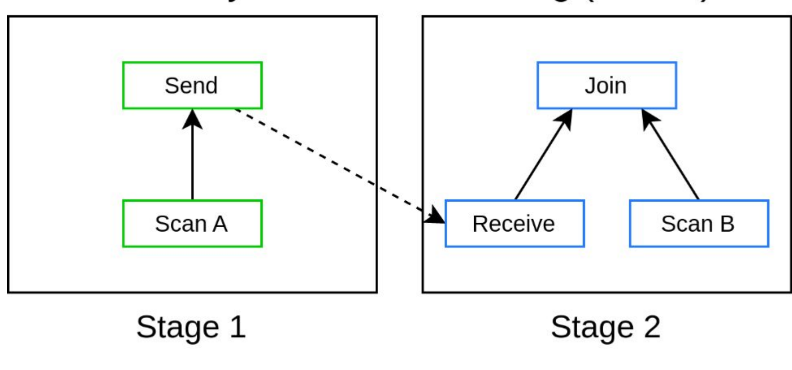
См. ConnectorPlanOptimizerProvider (Presto), ConnectorMetadata (Trino)

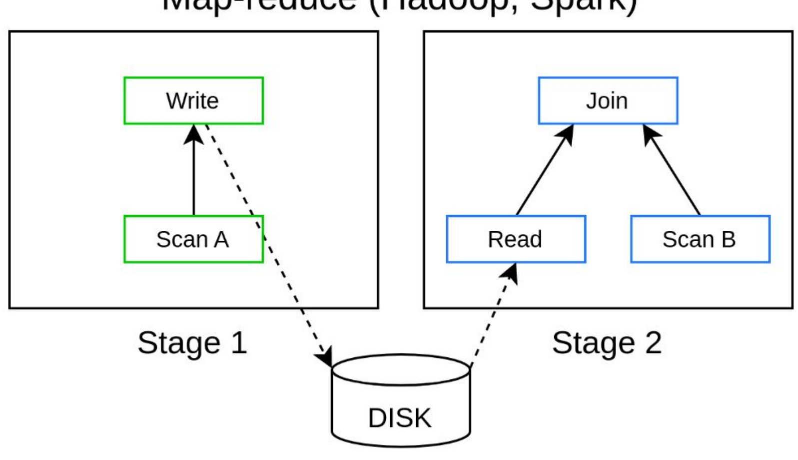
Выполнение: stages



- Координатор разделяет логический план на последовательность **stages**.
- Stage — это последовательность операторов, которые могут быть выполнены локально на воркере.

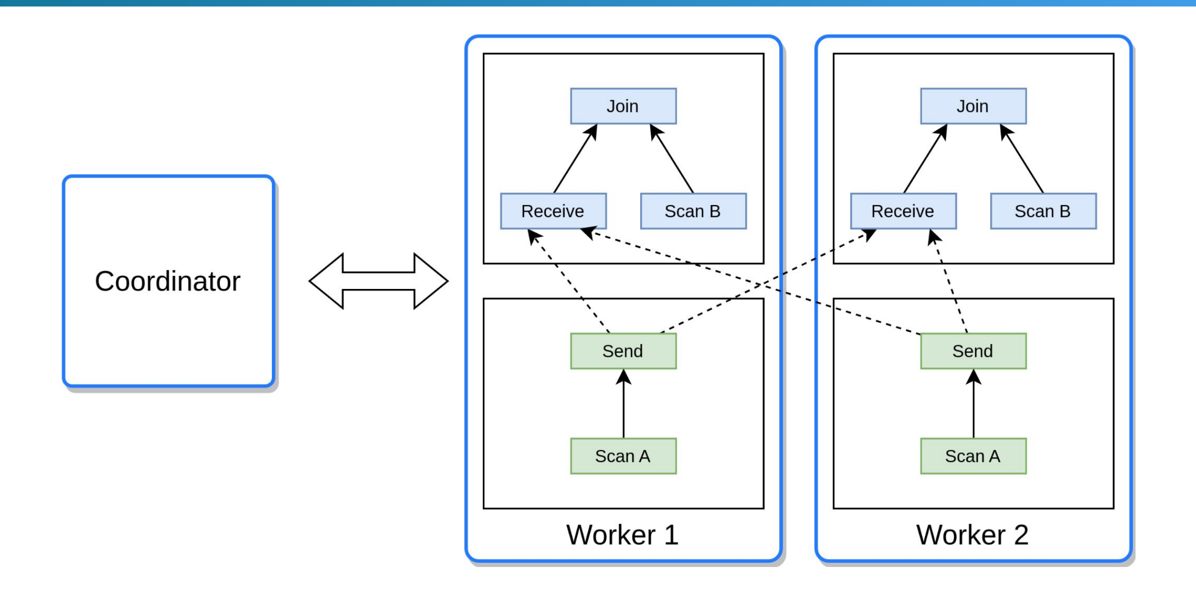
Выполнение: MPP vs MR





- MPP — данные передаются по сети, хранятся в памяти.
  - Быстро, но может не хватить памяти; хуже отказоустойчивость.
- MR — данные передаются через персистентное хранилище.
  - Медленнее, но можно обрабатывать очень большие объемы.
- Presto — это MPP с зачатками MR:
  - Умеет сбрасывать данные на диск, но не делает этого по умолчанию.
  - Инициатива Presto-on-Spark.

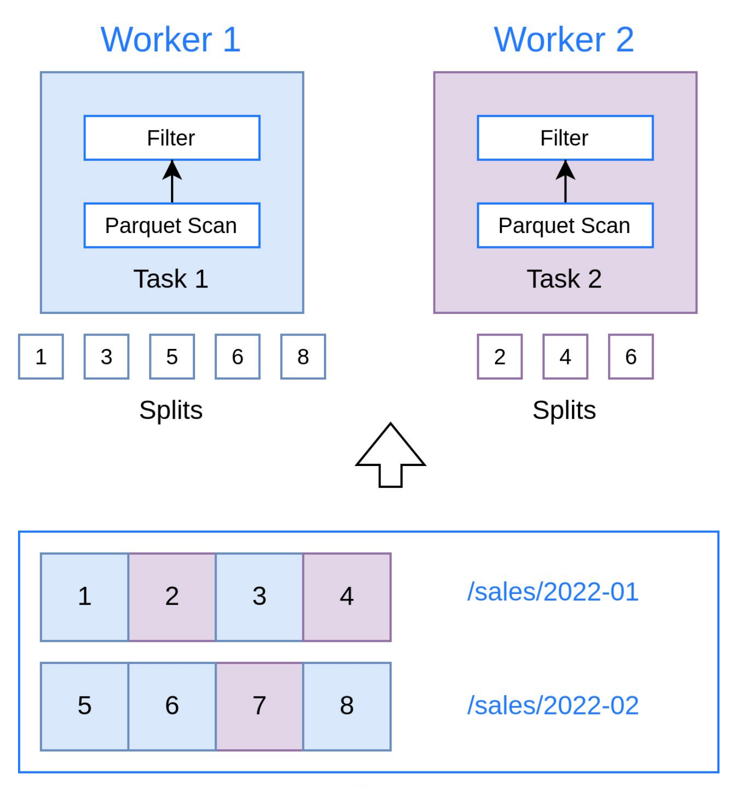
Выполнение: Task



Stage — это “шаблон” фрагмента запроса. У каждого Stage есть набор источников данных.

Task — это инстанс stage на конкретном узле. Координатор определяет, на каких узлах выполнять stage в зависимости от требований источников данных

Выполнение: Split

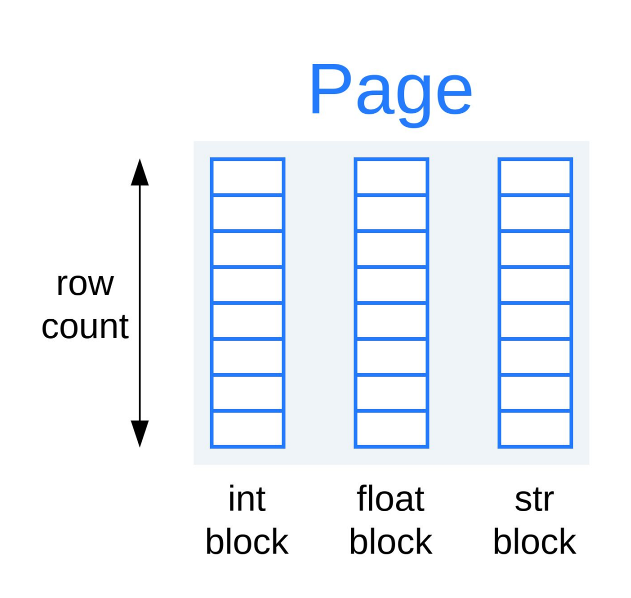


Split — это часть данных источника. Каждый источник данных представлен конечной последовательностью Split'ов.

- Таблицы Hive разбиваются по файлам и частям файлов.
- JDBC-источники всегда состоят из одного сплита, представляющего собой result set выполнения запроса.

См. ConnectorSplitManager, ConnectorSplit.

Выполнение: Page



Операторы получают и производят **Page** — набор кортежей в **колоночном** формате.

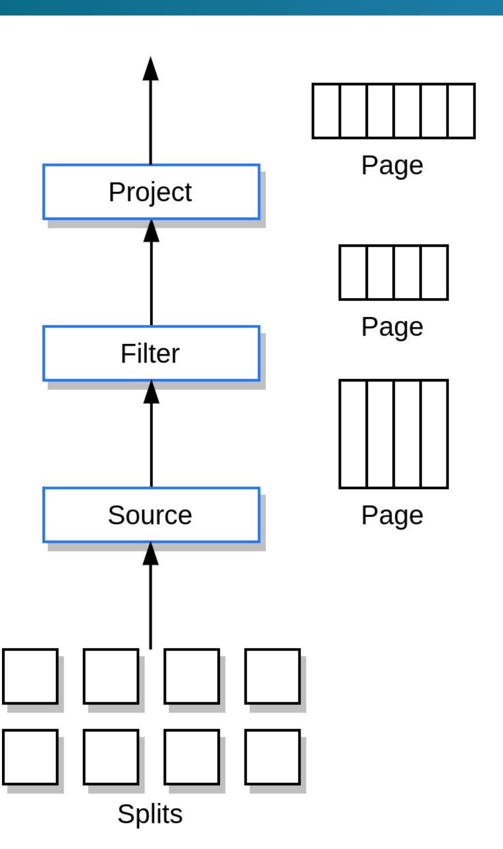
Каждый атрибут представлен объектом **Block**, который полностью инкапсулирует доступ к данным по индексу.

Пример `IntArrayBlock`:

- `int[]` — массив значений.
- `bool[]` — маркер NULL.

См. Page, Block.

Выполнение: push



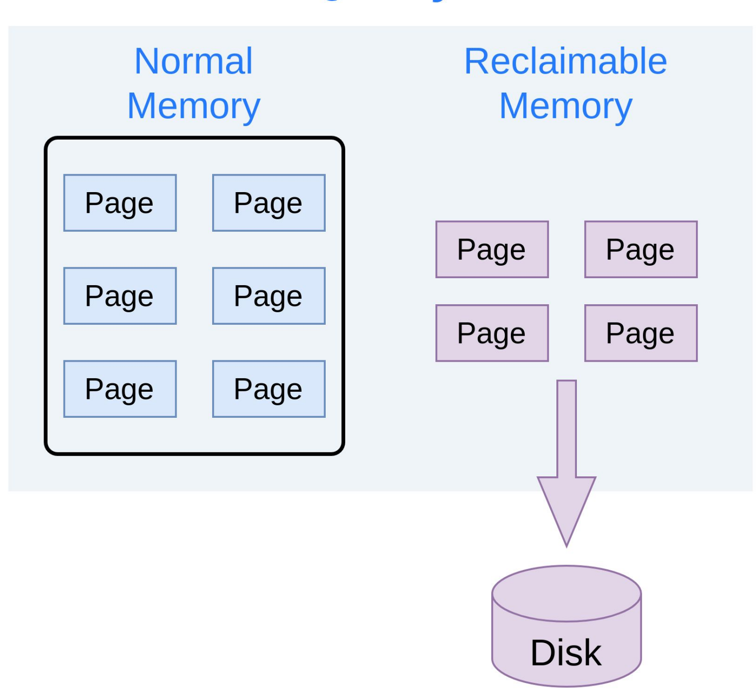
Presto реализует неблокирующую **push-модель** выполнения:

- Source-оператор получает split(ы).
- Оператор производит Page.
- Page передается следующему оператору (push).

Создание и поглощение page’ей отделено друг от друга:

- Оператор может быть не готов произвести Page. Например, Sort еще не получил все входные данные.
- Оператор может быть не готов принять Page. Например, Hash Join не принимает данные из probe-стороны, пока не получены все данные из build-стороны.

Выполнение: управление памятью



Многоуровневое управление памятью:

- Лимит на уровне узла.
- Лимит на уровне запроса.

Операторы используют `Page` не только для обмена данными, но и для внутреннего состояния:

- Хэш-таблицы.
- Сортированные структуры.

Spilling:

- По умолчанию при достижении лимита памяти запрос будет отменен.
- Операторы могут запрашивать память сверх доступного для запроса лимита. При необходимости эта память будет освобождена путем выгрузки данных на диск.

Выполнение: компиляция

HashAggregation

```
for row in page:
  // Найти группу по ключу
  int hash = hash(row)
  Group rowGroup = null
  for group in groups[hash]:
    if equal(group, row):
      rowGroup = group
      break

  // Обновить агрегаты
  rowGroup.consume(row)
```

hash (interpreted)

```
int hash = 0;
for idx : columns:
  if type[idx] is Int:
    hash = intHash(hash, row, idx);
  else if value is String:
    hash = strHash(hash, row, idx);
  ...
```

hash (compiled)

```
hash = strHash(hash, row, 2);
hash = intHash(hash, row, 3);
...
```

Интерпретируемый код вносит накладные расходы:

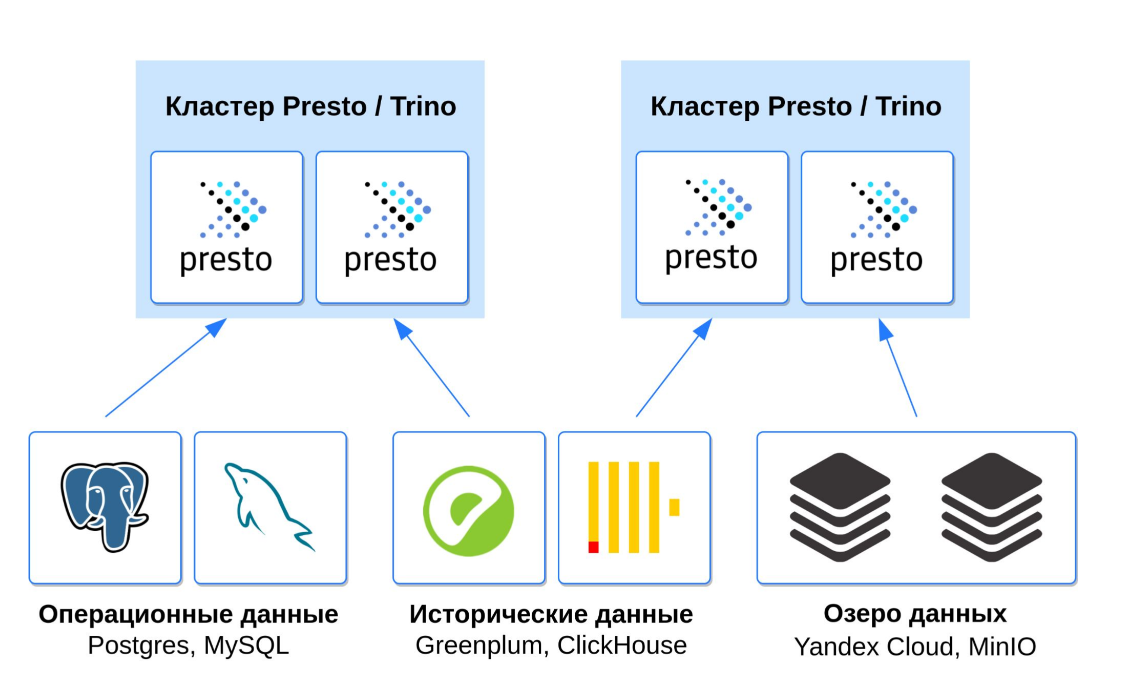
- Циклы.
- Условные переходы.
- Виртуальные вызовы.

Скомпилированный код позволяет избавиться от значительной части накладных расходов.

Выполнение: компиляция

- Операторы содержат указатели на функции. Например:
  - `Projection/Filter` – вычисление выражений.
  - `Aggregation` – вычисление аккумуляторов.
  - `Join` – вычисление фильтров.
- Функции компилируются с помощью ASM в процессе выполнения запроса, используя информацию о конкретном операторе.

Использование Presto



- Подключение к любому S3-совместимому object storage через S3 Hadoop FileSystem.
- Подключение к другим источникам данных через JDBC.
- Объединение данных из задействованных источников.

Итого

- **Lakehouse** — это набор подходов и технологий для совместной обработки больших объемов данных в файловых хранилищах из разных движков.
- **Presto/Trino** — это распределенная MPP-система, которая выполняет SQL-запросы, но не хранит данные (shared storage).
- Presto/Trino имеет коннекторы к ключевым системам и технологиям lakehouse-стека и не только:
  - HDFS и S3.
  - ORC и Parquet.
  - Hive Metastore и Apache Iceberg.
  - OLAP: Greenplumn, ClickHouse.
  - OLTP: Postgres, MySQL.
- Попробовать:
  - Presto: https://prestodb.io/
  - Trino: https://trino.io/
  - CedrusData: https://www.cedrusdata.ru/

Обратная связь
и комментарии по
докладу по ссылке



